Department of Mechanical Engineering

School of Engineering

University of Thessaly

Dimitris Rizopoulos' diploma thesis on

"Multi modal route planning in public transit networks"

Volos, Greece

September 2017

Supervising professor and thesis advisor: Dr. Georgios K.D. Saharidis

Members of the Selection board:

First member: Dr. Georgios K.D. Saharidis

Assistant professor at the Department of Mechanical Engineering, University of Thessaly

Second member: Dr. Georgios Liberopoulos

Professor at the Department of Mechanical Engineering, University of Thessaly

Third Member: Dr. Dimitrios Pantelis

Associate professor at the Department of Mechanical Engineering, University of Thessaly

The approval of this thesis by the Department of Mechanical Engineering of the School of Engineering of the University of Thessaly does not suggest the acceptance of the views depicted in this thesis by the author.

Credits and acknowledgements

First of all, I would like to thank my supervising professor and mentor Dr. Saharidis, not only for helping me go through the process of writing this thesis but also for giving me the chance to work with him for the last 3 years and years to come.

I would like to thank all of the professors at the Department of Mechanical Engineering and especially the professors at the Division of Production Management & Industrial Administration, for teaching me how to approach and work on bigger problems.

Finally, I would like to thank my family for supporting me throughout my pursuits. There are no systematic means of communication to describe how grateful I am to have Apostolis, Evangelia and Kiriaki into my life.

“Some people see the glass half full. Other see it half empty. I see a glass that’s twice as big as it needs to be.”

- George Carlin

“Multi modal route planning in public transport networks”

Rizopoulos Dimitris

Supervising professor: Dr. Georgios K.D. Saharidis

Volos

September 2017

Abstract

The current thesis analyzes in depth the problem of Multi-Modal Route Planning (MMRP) in public transportation networks. In the first chapters of this thesis, we describe more generic cases of route planning problems, corresponding approaches and, progressively, in the last chapters we focus on methods that aim to solve the multi modal case of route planning for public transport networks. The author also describes practical aspects of the methods, such as the homogenization of the data and the size of the problems that is generated in terms of computer memory and data structures. Specifically, in the first chapter we provide an introduction to the subject of the thesis and in the second and third chapters we discuss the history and the lastest advancements in route planning. In Section 4, we elaborate on the specifications that have been established for homogenizing and sharing MMRP data. Furthermore, in chapter 5 we present different approaches that we have developed for the problem of MMRP and are based on different fields of applied mathematics and computer science. Finally, in chapter 6, we discuss the results, their meaning and why they can lead to new perspectives on route planning problems. Sections 7 and 8 are the Bibliography and Appendix correspondingly.

Table of Contents

Table of figures

1 Introduction

In the mist of the economic crisis, many would doubt that we live in an age of profusion. However, if we take a look at the bigger picture there is more of everything nowadays. More computers, more cities, more nations, more goods, more services, and most importantly, more people. The world's economy has increased five times since 1950 and by 2050 there will four times more people on earth since 1950. We live in a vast world that is expanding by the second. Of course, all of the above statements can be viewed as positive changes in the world, since they allow to each one of us to have more choices, and respectively more freedom. This is the case with transportation also. In the last hundred years, there has been a moblity revolution allowing people and goods to move around a lot more than any other time in world history. This, of course, is a result of the developments in technology happening across all those years. Transportation has become faster and cheaper making life easier in many ways. However, those positive changes have sideffects. All those advancements in every day life led us to neglect many problems that have been created as a result of the excessive use of certain technologies.

The most important problem is the pollution of the environment caused by transportation. As reffered to in Wikipedia [1], the trasportation sector is accounted for an estimated 30% of the annual Greenhouse Gas emissions worldwide. The greenhouse effect, in turn, raises the temperature of the planet, disrupting its stability. Also, other kinds of pollution are caused by the transportation sector, such as noise pollution and carbon monoxide emissions, that affect in a bad way, and in most cases indirectly, the health of humans and animals on the planet.

While such problems arise, and in combination with the fact that more choices lead to more complexity, the need for better decisions when it comes to how we actually transport commodities and people from one place to another is imperative. These kinds of decisions are the answers to the problems that we deal with in the field of route planning. The problems, though, depend on the goal that a person sets every time. As we will discuss further in the main body of the thesis, different groups of people have different goals when it comes to transportation, and thus, several kinds of *questions* need to be answered in order for those groups of people to succesfully achieve their goals. Over the years, those questions and the decisions that serve as answers, have been grouped under categories, which we call *problems* in transportation sector and are usually solved with applied mathematics and computer science techniques. The Vehicle Routing Problem and the Travelling Salesman Problem are some of the fundamental problems that need to be efficiently solved and then apply the results to real world problems.

Better route planning tools lead to better optimization of the decisions and, in respect, more meaningfull transportation. In context of the current thesis, we will employ several techniques of applied mathematics and computer science in order to experiment on how a person or an organization can make decisions that will serve their goals better, while those decisions will not affect in an unwanted way other systems, such as the environmental eco-systems or economic ones.

2 Route plannning: Modelling and solution approaches

In this chapter, we are going to present the tools and methods that have been developed by the scientific community and industry specialists in order to deal with route planning problems. At first, we will try to give an overview of the field and as our analysis progresses, we will try to point out the important parts of theory and the terminology that we want the reader to understand better. Then, we will move on to Section 2.1 and 2.2, where we will present a review of how the different problems are modelled and what methodologies can be used to derive solutions to the models.

To begin with, let's make clear the distinction between transportation planning and route planning. The former refers to designing and assessment of transport facilities, whereas the second, which is the field that this thesis is about, refers to the calculation of optimal, or sub-optimal, ways to get from a point A on a map to a point B. Of course, A and B represent physical locations on the planet and the object to be routed between the two points is not always a person, but can be a vehicle or commodities. In general, there is a variety of instances of problems that fall under, or are associated with, the term of route planning. For example, one of the most widely dealt with problems is the problem of *the shortest path problem* [2]. As it is obvious by its name, in the shortest path problem the modeler/researcher has to find a systematic way to come up with the paths in the network that are the shortest or have the minimal relative cost, compared to other choices. In other words, while in general route planning, the tools or solutions, are calculated based on "optimality", which is a generic term, in the shortest path instance, we specifically search for routes that their optimality is directly dependent on some cost. The most common case is when the costs refers to distance between locations and the shortest path refers to minimal distance to get from one point to another. That is not the case with other problems though. For example, in the *Vehicle Routing Problem* (VRP) there are more than one criteria and constraints that one should consider in order to derive an optimal path. Thus, optimality is not only dependent on minimizing a single distance between points. In a VRP problem, and at its most common variation, one would need to calculate several optimal paths for fleets of vehicles that need to deliver products at several storage facilities [3]. An important constraint that is present in such cases is the time-windows constraints. Not all facilities are open 24 hours per day and 7 days per week, and as a result every time we calculate a path that a truck follows, it has to comply to these restrictions.

Another more generic instance of the VRP problem is the *Travelling Salesman Problem* (TSP) [4]. The TSP can also be viewed as a hybrid of the shortest path problem and the VRP because the path calculated as a solution to it, is the shortest path that visits or covers a set of points (what we call facilities in VRP). Specifically, in the TSP we have to calculate an optimal path for a salesman to follow, where he or she needs to visit a set of nodes of the network, covering the minimal distance. After his trip is finished, the salesman has to return to the starting position. However, the salesman is not bound by any other constraints in the TSP, as we consider the facilities/nodes of the network to be theoretically available at all times. A generalization of the TSP is the Travelling Purchaser Problem, in which the traveler needs to follow a route and purchase some goods at the best prices available. Again, in the generic versions of those problems, the travelling salesman is not bound by any kind of constraint.

Other variants of route planning problems are the *Range Problem* and the *Earliest arrival problem*. We chose to mention those together because of their time dependent nature. The

first, the *Range Problem*, deal with planning route along networks that contain time-related constraints, that are based on a range of time, within which routes from an origin to a destination are accepted. Routes calculated outside that range are unacceptable. It is a similar notion to time-windows used in other routing problems. The second, called the *Earliest Arrival Problem*, is a type of problem, where we simply use instead of *distance* units as weight for our decisions, we try to minimize the *time* it takes to go from starting to ending point. At first, one may think that the cost in time units and the cost in distance is the same, but in many occasions, they differ greatly. Now that we have covered some basic principles about our subject we move on to Sections 2.1 and 2.2, in which we will cover the modelling approaches and solution algorithms.

2.1 Modeling of the problem

So far, we have described the basic instances of route planning problems. Those kinds of problems lay at the foundation of every navigation system that is used in the real world, ranging from software that aids drivers to cut through traffic in a big city or professionals and companies who need to make the transportation of goods faster and cheaper. Before we move on to the complicated case of *multi-modal route planning*, it would be wise to present the basic tools that have been developed worldwide to deal with those problems. As reported in the literature review by Bast et al. [5], graphs and graph algorithms have been the approach that has dominated alternative approaches when it comes to shortest paths. Graphs themselves are mathematical structures that are used to represent the relations between a set of objects. Their applications range from mathematics to academic disciplines such as psychology and sociology. The components of graphs are sets of Vertices and sets of Edges, with former being represented by *V* and the latter represented by *E.* In the context of the current thesis, we represent a graph structure with *G (V, E)*. When graph *G* is used for the route planning case, its vertices represent the locations or facilities that the object to be navigated will possibly visit or use and the edges *E* represent the connection from one vertex to another. Usually, according to some criteria those connections are related to a cost.

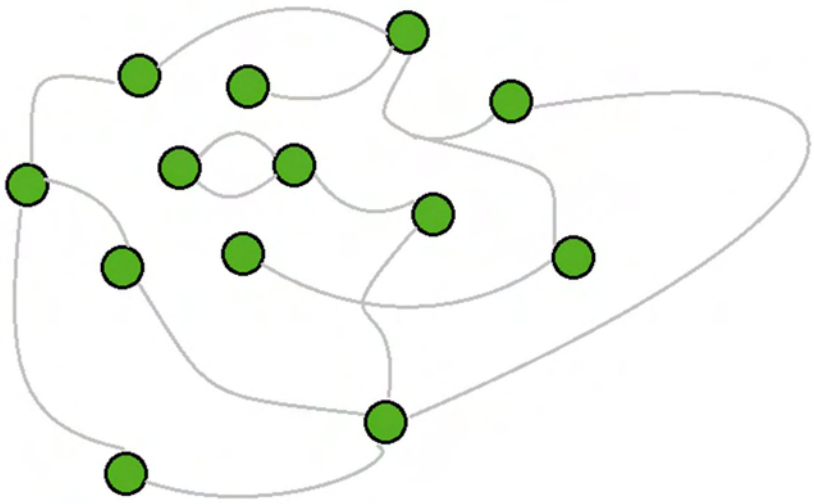

Figure 1: A graph G (V, E) that can be used to represent a transportation network. In this case the green dots or vertices can represent different cities and the grey lines or edges can represent the relationship between the vertices, which is the distance in most cases. Another example, which is totally different from route planning, could be a network of people, with each individual being represented by the vertices and the relationship between them (friends, co-workers, family, etc.) being represented by the edges.

The reason for using graphs in route planning is because of their robustness and flexibility. They have been proved to work well for a large number of applications and different kinds of datasets. Graphs can be used for small applications up to huge navigation systems with intercontinental coverage. As mentioned in the work of Bast et al. [5], the researchers have tested them against enormous datasets that produced graphs with 18 million vertices and 42.5 million directed arcs (edges). Of course, variations exist in their performance. There are many different ways to implement them and many ways to search for optimal solutions in them. *Graphs algorithms* are the *search tools* used on the graphs in order to get to the best combination of segments that lead from the starting vertex to the ending one. Before continuing on the explanation of other techniques, we would like to remind to the reader that graphs are very abstract notions. As mentioned before, they are used in many fields that are not related to computer science or route planning. However, in the context of this thesis, when we refer to an approach as a *graph approach*, we refer to approaches that are based on *graph data structures,* meaning that the implementers of the approach, create structures in the memory of the computer that work like graphs and allow faster searches due to specific attributes.

Next, let's make a reference to Linear Programming (LP) and Mixed-Integer Linear Programming (MILP). Those two methods of applied mathematics are used in a big variety of applications. MILP is mainly used in route planning because it is considered as the go-to approach for VRP problems. The idea of a MILP approach towards a VRP was first introduced more than 50 years ago [6]. MILP modelling techniques allowed researchers and industry experts to take better management decisions when it comes to vehicle routing. In its essence, MILP is more flexible by allowing the modeler to represent decisions as mathematical entities and express the constraints and relations between those decisions by mathematical equations, equalities and inequalities.

Moving to other types of problems, although recently there has been conducted some relative research by Saharidis et al. [7], there has been no extensive research of possible outcomes of MILP formulations of route planning on public transit networks. That is not the case though with applications of MILP in TSP problems. While there are many variations and many ways to model the TSP, the most common approach to such problems is MILP. As referred to in the work of Saharidis et al. [8], the TSP is an assignment problem, where each connection is represented by a decision variable, while we make sure that we introduce to the model the *sub-tour elimina*tion constraints. By the way, sub-tour elimination constraints are also present in the work that we will present later on, and are a common underlying trait of the TSP, VRP and MMRP, when modelled with LP or MILP.

Considering the taxonomy of the approaches, all of the techniques used in the field of Optimization, and those that will be developed and referred to in the thesis, fall under one of three categories of approaches. Those categories are exact approaches, heuristic approaches or hybrid ones. The main criterion that distinguishes one approach from another is whether that approach searches extensively all candidate solutions to a problem or not. On the one

hand, we will build *heuristic approaches* when we want approximate solutions or calculate approximations of solutions. What that means is, that the answer that we calculate for the problem that is under research is not the absolutely optimal one, but it is close to it. In other words, we use heuristic methods because the application of the solution in the real world, is not required to be globally optimal, meaning that a gap between the heuristic solution and the globally optimal is not a problem for that case. Also, let's point out that when the solution is an approximation of the solution, we know it is not the optimal one and we know the gap we need to cover to get to the optimal, whereas when calculating an approximate solution the gap is unknown. On the other hand, there are the *exact approaches*, where the gap between the solution found by the exact approach and the optimal solution of the problem is zero. It is important that this statement can be proved by the way that the approach is built. Although, exact approaches tend to provide us with better solutions in terms of reduction of cost, generally speaking, they cannot yield solutions in the amount of time that heuristics can, and usually take much more time to complete the calculations. In the MMRP, graph and MILP are heuristic and exact methods correspondingly. That means that later on in the current thesis we will explore several advantages and deficiencies of the methods and see hands-on examples on how exact and heuristic differ from each other. Finally, we will present *hybrid approaches* which are a combination of the above perspectives. We will try to incorporate advantages of both methods into one.

But, before getting into the actual presentation of the modelling approaches, let's make a reference to *precomputation techniques* used in most heuristic methods. Like all of the aforementioned terms, introducing the reader to precomputation techniques, will aid us to better understand the next chapter of the thesis. With no further ado, precomputations are techniques that are applied in an "offline" stage of the process of calculating results, and enable us to speed up the same process later on when we need to. This is especially useful when we need a navigation system to be the back-end of an online web-platform. In such cases, usually there is a graph builder module that does all the precomputations and data structure building when the system is in "preparation" mode or offline. Then another part of the application handles the online phase of the system and answers to queries of the user's browser. That is the case with OpenTripPlanner (OTP) [9], an open source library used to build such systems. OTP uses some of the methods we will describe in the following paragraphs in order to turn vast graphs into smaller ones that can be managed by the computer. The gains from using precomputations techniques, is not only computer memory (RAM), hard disk space and CPU usage but also speed in the final stages of computations. Nonetheless, we should mention that most of the graph approaches are, in one way or another, connected to Dijkstra's algorithm, an algorithm first introduced in 1956 by computer scientist Edsger W. Dijkstra. Although Dijkstra in his initial proposition, described an algorithm that was not associated with precomputations, through the years many variations came up and, finally, the ones that become most well-known are the ones that use heap space to order the nodes by some criteria and guide the search. We mention that because we want the reader to get to understand that graph algorithms, and generally approaches with graphs, have always been connected with some type of precomputation even on a theoretical level. Later on in the thesis, we provide more insights to the way that Dijkstra's algorithm works. OTP is also used by the *GreenYourMove* project [10] which builds a pan-European journey planner and helps consolidate modern policies related to transportation all over the EU.

Figure 2: The OpenTripPlanner logo. OpenTripPlanner has been for many years now the go-to open source software for companies and organizations that want to set up regional journey planning systems. Instances of OpenTripPlanner's software version are running all over the world, serving as the backbone of many platforms, aiding people in their everyday lives. For a list of deployments of OpenTripPlanner please check the link at [11].

2.1.1 Graph modeling techniques

Considering the graph approaches that have been developed in the past, one can claim that the modeler has many options for how he or she will approach the problem. One of the most researched topics and, for many years, mainstream way to model a route planning problem is the Contraction Hierarchies (CH) method, where the nodes of the network represent physical locations and are classified according to some criteria into different layers of nodes, creating a *multi-layered graph*. After their accession to layers, they are "Contracted", meaning that nodes that are not important, and their corresponding connections, are removed and replaced by shortcuts. In this way, the search algorithms can focus on finding routes on the higher layers of the network and then easily reduce the result to all the necessary nodes in the less important and contracted layers. As described in the journal article by Geisberger et al. [12], the CH method requires an ordering of the nodes in the network, which allows big graphs to be searched more easily by the CPU, because the search algorithm does not need to search every single option that is available but rather search for optimal solutions based on the layer that each node belongs. The initial ordering of the nodes and the accession are based on bidirectional queries that run from all the nodes to all the nodes, measuring the importance of the node every time.

Of course, as it is obvious by now, this modelling technique is a heuristic method, thus the resulting path is not guaranteed to be optimal. CH is one of the main approaches that has been used by OTP for most of its stable versions throughout the years.

Another set of techniques used for modelling a graph representation of a transportation network is called Arc Flags (AF) [13]. How it works is that, after creating an initial graph, similarly to CH, it partitions the nodes of the graph into K-cells that are the same

in size in terms of the number of nodes that they contain. Then, it assigns a K-bit flag to each node of the network, with each bit taking the values of 0 or 1 if the closest node to the one that the flag is on, is in the Kth cell. Similarly, to this technique we developed the *Beehive* partitioning technique, that will be introduced in later Sections.

Moving on, another technique similar to the ones above that has been successfully to the problem is the Geometric Containers (GC) method [14]. GC reduces the search space by exploiting geometric information about the initial graph structure. GC, as introduced by Wagner et al., is a framework that the pruning of the search space happens according to some parameters that the modeler decides. As a general rule, a precomputation based on GC will have the arcs or edges of the graph labelled (instead of nodes in previous techniques) with a label *L(a)*, which contains all the nodes to which the shortest path starts from *L(a)*. So, for every pair of nodes that we want to find the shortest path, we can prune the arcs that do not contain the target node. Notice that for a label for an *arc a,* there will be multiple nodes assigned and maybe in order to calculate the path from point *X* to *Y* we need to calculate optimal paths for sub tours for points between *X* and *Y.* GC is one of the methods with the best results presented in the literature, however one needs to consider that takes the most time for precomputation because it needs to precompute every path before-hand according to geometric characteristics of the graph (e.g. coordinates of nodes on a geographical map).

Transit Node Routing (TNR) [15], is, as well, a widely used approach known for its good results. The inspiration for its creation comes from a common conception that every time that we travel towards a destination we follow the same basic route. For example, for a trip from Athens to Thessaloniki, for the most part of the trip, you would follow the same route that we would follow if we went from Athens to a set of other cities like Larisa Volos, Veria or Katerini. Now if we consider the road network as a graph, the idea mentioned above means that the route that we would follow would contain partly the same nodes. Out of those same nodes there are some that are of greater importance and some of lesser. The vertices that correspond to locations of greater importance, like the Maliakos Gulf or the Tempi passage, are most likely to be in those trips and are considered nodes of the higher priority. In more depth, the TNR approach, makes use of those higher priority nodes. Just like in CH, a multilayered graph is created for the network. Then the top layer is used as the Transit Node set and is composed out of those higher priority nodes. Not only the searches are executed based on the set, but most of the shortest paths between the vertices of the graph that belong in this Transit Node set are precomputed and serialized to the hard drive for permanent storage. Of course, large graphs can get very sparse and the routing happens by exploiting some locality filters that exclude some far away nodes from precomputation. The search for an optimal path can then be performed with a few look-up steps when needed with TNR.

Finally, worth mentioning are the *compressed database* techniques, where all calculations are made and stored to a database, after they are compressed, in order to take up less space. What exactly is stored in the database is, for every possible starting node of the network and every possible corresponding destination node, the exact path that an object to be navigated would need to follow in order to get there. Associative arrays (dictionaries) can be used as the suitable data structure for such applications, so that the pair of nodes themselves can be stored in the database, in correspondence with the paths to follow each time. This, as one could easily understand, is the most *brute force* way of precomputation. After precomputing paths for every possible selection of starting and ending points, only a minimal search is required in order to find an answer to a problem. In addition to *compressed database techniques*, those methods come under the name *table-lookup* techniques.

2.1.2 Mixed Integer-Linear techniques

Now, considering the MILP methods that one can use to model a route planning problem and the underlying network, the cases of problems that MILP techniques are mostly used are the VRP, TSP and variants.

MILP and LP techniques fall under the umbrella of Optimization techniques. Optimization is a sub-field of applied mathematics and includes practices that have been proved to be beneficial and lead to better management decisions. Although, it contains many other important subfields, like *Stochastic programming* or *Non-linear programming*, MILP and LP approaches have been extensively used in many fields. Their advantage over other practices is that problems that in the most part contain linear mathematical expressions are easier to solve by algorithms that search their solution space for optimal solutions. On the contrary, their disadvantage is that linear expressions can't be used for every problem and many real-world applications need more complex mathematical structures to be properly modelled.

Independently of the specific type of Optimization problem, every program needs to have three components. The *objective function*, the *constraints* and the *decision variables*. The layout of the whole problem is mostly affected by the decision variables that we choose, that are the mathematical objects that are used to represent the final decisions. Their values are the answer to the problem. Then according to the decision variables, we model the constraints which are a set of rules that the program needs to follow. The objective function is used to model the criteria that according to which the optimal solution is picked. Those are very important elements that will be integral parts of our proposed work. Although, we briefly review them here, we will explain them in more depth in Section 5. Moving on, we will provide information on the modeling of route planning problems.

In the VRP, by having a set of depots and the connections between them, we can then use mathematical expressions to represent every decision and the relationship between them. First introduced in 1959, there has been extensive research in this field. The assignment of multiple trucks to facilities and the need to take under consideration multiple criteria, favored the use of MILP and exact methods. Other heuristic and metaheuristic methods have been applied, like Genetic algorithms and Tabu search algorithms with success. Although the VRP is outside the scope of the current thesis, it can provide the reader with great insights about MILP. Also, VRP variations span from applications with environmental criteria to real world case studies that can provide a wide perspective on the subject. For that reason, we encourage the readers that want to get to know more about VRP formulations to read the thesis of Konstantinos P. Chatziliadis under the title "The Vehicle Routing Problem with Time Windows A case study", which was conducted 2016 in the department of Mechanical Engineering of University of Thessaly.

On the contrary to VRP applications, there are not many examples that we can refer to where the modeler chooses to model the problem and the data of a route planning problem when single objects needed to be routed. In such simpler cases, where there is no need to take under consideration quantities, assignments and the problem to be solved is a Shortest Path problem, then graph algorithms seem to be more suitable to do the job.

2.2 Solution algorithms to route planning problems

2.2.1 Solution algorithms for graph modeling techniques

Over the years, many algorithms have been proposed on how to search for optimal solutions in graphs. In most cases, the solution that we seek is the shortest connection between vertices. Cases where *arcs* and *vertices* have other attributes, such as labels, and may have other criteria according to which a graph should be searched for optimality will be ignored for now.

Despite the fact that it was introduced over 50 years ago, the algorithm that has served as a panacea for most types of problems is the algorithm of Dijkstra *[16]*. Its applications vary from biology, and problems such as the finding of the network model in the spreading of an infectious disease, to research of internet routing protocols and phycology. Dijkstra, the creator of the algorithm, pointed out that "simplicity is the prerequisite for reliability". He used this phrase as his philosophy and this quote has been replicated and shared by many people over the last decades. Similarly, the algorithm's function is based on that philosophy and can be described in 6 simple steps. When those steps are followed recursively, the algorithm returns a shortest path tree from a starting node in the graph to all the other nodes. This simple "recipe" that we will present below, allowed many researchers to base their work on it and make the necessary modifications in order to comply with the needs of their problems. Before moving on to specify some alternatives to the basic version, we encourage the reader to check

Figure 3 for the exact steps of the algorithm.

Dijkstra's algorithm 6-step process, given A to be the starting point and B be the ending point:

1. **Step one**: Assign to all nodes, except for A, a "to-reach" cost equal to Infinity. The "to-reach" cost for A should be 0. For each node in the graph assign a label, containing the shortest path to the node in the form of a list with a set of nodes in a specific order.

2. **Step two**: Set node A as the "current node" and all other nodes as unvisited nodes and place them in the "unvisited set".

3. **Step three**: For the "current node", consider all of the neighbors and calculate all of their "to-reach" costs. If the newly calculated "to-reach" cost is smaller than the previously existing one, assign the new cost to neighboring node, and update the label with the shortest path to the node, which will of course contain the current node that is under investigation.

   We provide an example for how step three should escalate. Suppose we have node X that is the current and a set of neighboring nodes Yi, with ranging from 0 to a positive integer N. From the previous iteration, we have

a “to-reach” cost assigned to all nodes including X and all Yi, which corresponds to the cost of the shortest path from the starting node A to each node. Next, let’s consider that the “to-reach” of X is 5, then we find the cost from X to every Yi and add it to a set of “to-reach” costs Xi, for i in the range 0 to N. If some of the Xi for every one of the neighboring nodes Yi is lower than the “to-reach” costs of the Yi we update them. Otherwise, keep the current value.

4. **Step four:** When all of the neighboring nodes are checked, remove the current node from “unvisited set”. The node will be considered “solved” and will never be checked again.

5. **Step five:** If the destination node B has been marked visited or the smallest “to-reach” distance is equal to infinity, meaning that there is no connection between A and B, then stop the algorithm as it has finished.

6. **Step six:** Else, select the unvisited (not neighboring) node with the smallest “to-reach” distance and set it as current node. Go to step 3.

When the final node has been extracted from the list of unvisited nodes, that means it is solved, we can check its label with the shortest path followed/to follow to the node.

Figure 3: Description of Dijkstra’s algorithm. (DISCLAIMER: Many versions have been introduced since 1959, so conflict versions to the one we present may be found. Our version is based on the webpage [17])

A* algorithm is another Dijkstra-variant algorithm that has been used successfully. It has been linked to better computational results and is superior to most other approaches because of the speed and low usage of CPU cycles that it allows. While in Dijkstra algorithm we based our search on a priority queue of nodes according to their distance from the starting node, in A*, we assign a “to-reach” cost to every node, too. This “to-reach” cost refers to the cost that is needed for the algorithm to connect the node to the destination node. As a result, the priority queue is constructed according to the sum of the two costs. Of course, this type of heuristic, allows the algorithm to perform a directed search that, at every step of the algorithm, allows to minimize the distance to the final node at greater speed, guiding the whole process to better choices more quickly. A* is also the algorithm that has been employed for many years by the creators of OTP. Other famous routing software libraries and organizations use A*, such as pgRouting [18] and GraphHopper [19] libraries.

Other Dijkstra-based algorithms that are used to search the space of the graph for available solutions exist. One of them is *bidirectional Dijkstra,* as it often referred to. In this case, simply, for a starting point A and an ending point B we execute two instances of Dijkstra or A*, one from A to B and another from B to A. The search, and consequently the algorithmic process, is terminated when both searches from A to B and from B to A have solved the same node. By *solved*, as referred to in Figure 3, we mean that all neighboring nodes of this particular vertex that has been reached by both searches, have been examined and the intermediate node has been excluded from the list of unvisited nodes. Then we connect the two optimal paths to the intermediate node, which has been solved by both algorithms, both the straightforward and the reverse search to produce the shortest path. Also, sometimes

referred to as *bidirectional search*, this algorithm can have different termination criteria. On the one hand, we can use the solved by Dijkstra criterion or the ending point of the algorithm can be when we have *solved* every vertex in the graph. In other implementations of the more generic bidirectional search, the method can stop when one of the two searches has reached the A or B and then according to some heuristic rules, the two searches are combined into a more optimal one.

Finally, the last algorithm that is Dijkstra-based is Bellman Ford algorithm. The result of its use is the same as if we used Dijkstra's algorithm, we get the shortest path from the starting to an ending point. On the contrary, the result of their execution is different, because Bellman Ford is a one-to-all algorithm, returning a list with multiple shortest paths from the first node to every other. While, can easily choose from the list the pair of nodes that we want to, Bellman-Ford algorithm proposed by Alfonso Shimbel in 1955, takes more time to do all those calculations. As for the calculations themselves, given a graph of N vertices, Bellman-Ford works in N-1 iterations. In every iteration, we get to examine every node and check the connections to neighboring nodes, keeping track of the smallest "to-reach" distances from the starting to each node. In every iteration, we skip the nodes that we haven't yet reached because they were not neighboring to our other nodes. Running the algorithm N-1 times guarantees that all nodes have been examined and shortest paths to every node have been recorded to labels.

2.2.2 Solution algorithms for Mixed Integer-Linear techniques

As an extend to Section 2.1.2, we will now present the algorithms that have been used in the literature in order to search the solution space of a MILP or LP formulation and calculate the optimal solution. Just like graph-based approaches, mathematical programming approaches we have to formulate the problem and make a representation for it in the computer memory. Then, we have to apply a search algorithm to that structure to exploit its properties and derive solutions. We would like to remind to the reader that when the formulation is that of a route planning problem, the *solution space* of the problem is all the paths that the object to be routed has available and the optimal one is, if distance is the only criterion we consider, the shortest path to its destination.

First and foremost, one of the most used algorithms is *Simplex algorithm*. But before we proceed to describe the algorithm, let's make a short reference to an important attribute of Linear Programs. As we have discussed, LPs are problems that can be expressed in the following canonical formulation:

Maximize	$c \cdot x$	(1)
Subject to	$A \cdot x <= b$	(2)
	$x \geq 0$	(3)

Where *c* and *b* are vectors of coefficients and *A* is a matrix of coefficients. The mathematical expression (1) is called the objective function of the model and its value serves as the criterion for which set of *x*'s is the optimal solution. The most important attribute that

the reader needs to understand is that (2) is a system of linear inequalities and in terms of geometry it defines polytope that can represent our solution space. For instance, if we try to solve a LP program with three *x* variables, then the solution space would be a three dimensional one and the polytope could be pictured for a certain set of constraints like in the Figure 4 below:

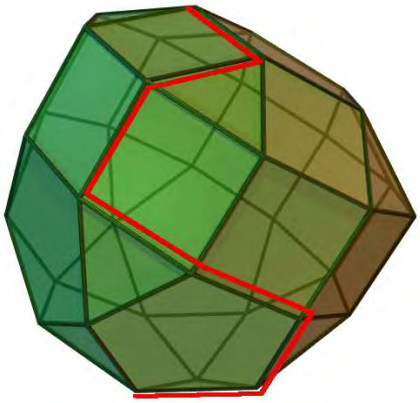

Figure 4: Solution space of a LP in 3-dimensional space. Image from source [20]. The red line represents the extreme points of the convex polytope that are evaluated against our objective function by the Simplex method.

For now, please ignore the red line on the polytope. Figure 4 is very important for our understanding of Simplex, because the algorithm works by exploiting this structure and checks only the edges of the geometric object. In particular, the algorithm starting from an initial vertex of the polytope, checks every vertex finding every time better and better solution, choosing the vertices to be checked according to if they are neighboring to the one under consideration and according to gradient of the polytope towards the vertex. Presenting the exact way that Simplex algorithm is outside the scope of the thesis, however, we encourage readers to take a look at the book of Hamdy A. Taha [21], which includes many LP and MILP topics, including the exact steps to implement Simplex algorithm.

Moving on, another approach is the *Dual Simplex algorithm*, which is based on Duality Theory. In short, the theorem points out that optimization problems can be viewed from two perspectives, one being the primal perspective and the other being the dual of the primal, or simply put, the *dual.* Duality theorem has a variety of applications in LP, but is very useful when the polytope of the solution space is considered convex. Then according to Karush-Kuhn-Tacker conditions, the gap between the solution of the dual and the primal is zero. In the case of Dual Simplex, we apply the same rules that we applied for the primal LP to its Dual LP, acquiring in the same manner the optimal solution. We epigraphically make a reference to the most important advantage of the dual simplex over the initial simplex, which is that is numerically more stable, because of the attribute of dual problems to be always feasible.

Other techniques used are the *ellipsoid method* and the *interior point methods*, which are used in more special cases of mathematical programming. Ellipsoid method is generally used for convex programs, and are very useful when they are applied on linear convex which we know that are feasible, they can come up with the optimal solution in a finite number of steps. Interior point methods are used on programs with non-linear convex polytopes as

feasible regions. For more information on the methods we suggest the reader to check the book by Dr. Taha [21].

Finally, the branch and bound (BnB) algorithm is an important solution approach that we will be using. Although, one could describe it as an algorithmic paradigm rather an algorithm that calculates the solutions themselves, BnB is based on the divide-and-conquer idea and enables us to systematically enumerate all possible solutions to a problem more efficiently. The main idea is that every solution approach that abides by the paradigm, considers the set of candidate solutions as a tree, which has as a *root* the full set of solutions and as *branches* has subset of that set. By exploring the branches, the algorithm is able to calculate solutions, which serve as bounds to the values of the starting full set of solutions. The algorithm stops when it cannot compute better solutions according to those bounds. In our case where we are trying to solve a MILP program with a convex solution space, each branch is linear relaxation of the starting problem and provides an upper or lower bound to the final value estimated by the method.

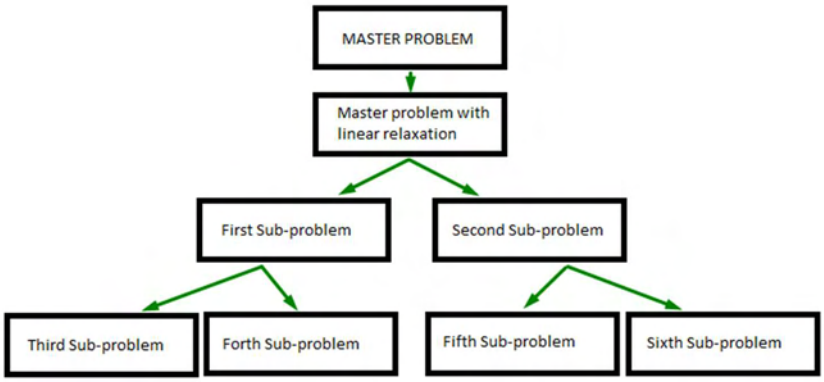

Figure 5: An example of the tree structure that is used by the BnB method

Branch and Bound is popular nowadays and is used by most of the LP and MILP solvers. In our approaches, we will use the COIN-OR solver [22], which is based on the BnB. Particularly, it uses a similar method called *Branch and Cut*, which incorporates the cutting planes technique. Another very popular solver that uses the BnB framework is CPLEX, developed by IBM Research [23] [24]. CPLEX is the optimizer with the best performance in the academia and industry.

3 Multi Modal Route Planning in Public Transport Networks

The multi modal route planning problem (MMRP), similarly to the route planning problem, is fairly new topic. It had its first practical application during the 1970s, where travel agents needed a way to manage flight tickets and aid travelers in their interchange between the different modes of transport, in order to get from their town of origin to their travel destination. Its popularity raised with the development of modern-day public transport services. In the 20th and 21th century, cities like New York, Los Angeles, Paris, London, Moscow and Beijing developed into what we call megacities. Transportation in those dense urban

environments was challenging and transport operators in their attempt to serve as many customers as possible, as well as, gain bigger financial profits, developed really complex services. That was a turning point for the development of such algorithms. The need for their use in everyday life by big numbers of people, gave a push to the academic and industrial communities to come up with ways to solve the problem faster. Also, another important role has played the lowering of the cost of travel and the increase in tourism. When travelers visit places, they don't know information about the transportation services available at the region, so they are in need of navigation services and guidance. All those practical changes in the way the world operates led to MMRP systems being in great demand and algorithms that can support such systems became sought after.

The basic difference of a MMRP problem in comparison to other Route Planning problems is the direct time-dependence of MMRP. The reason for that is that multi-modality in most cases refers to combination of public transport modes whose function is based on time-schedules. The cases where a public transport service does not run on a schedule is very unlikely. Only Taxi services could be considered such. So, while route planning generally solves problems that can either be time-dependent or not, MMRP focuses only on schedule based transportation. What that means from a theoretical standpoint is, that while in simple route planning one could have static data, in MMRP, that is not the case, because one would need to incorporate in the algorithms timetables that often change. For that reason, and as we will see later on in this thesis, MMRP algorithms needed to be more robust and powerful, consuming as little CPU cycles as possible. Again, most approaches have been graph- and Dijkstra-based, but as the years passed there have been approaches that are not based on graphs and come up with really different perspectives.

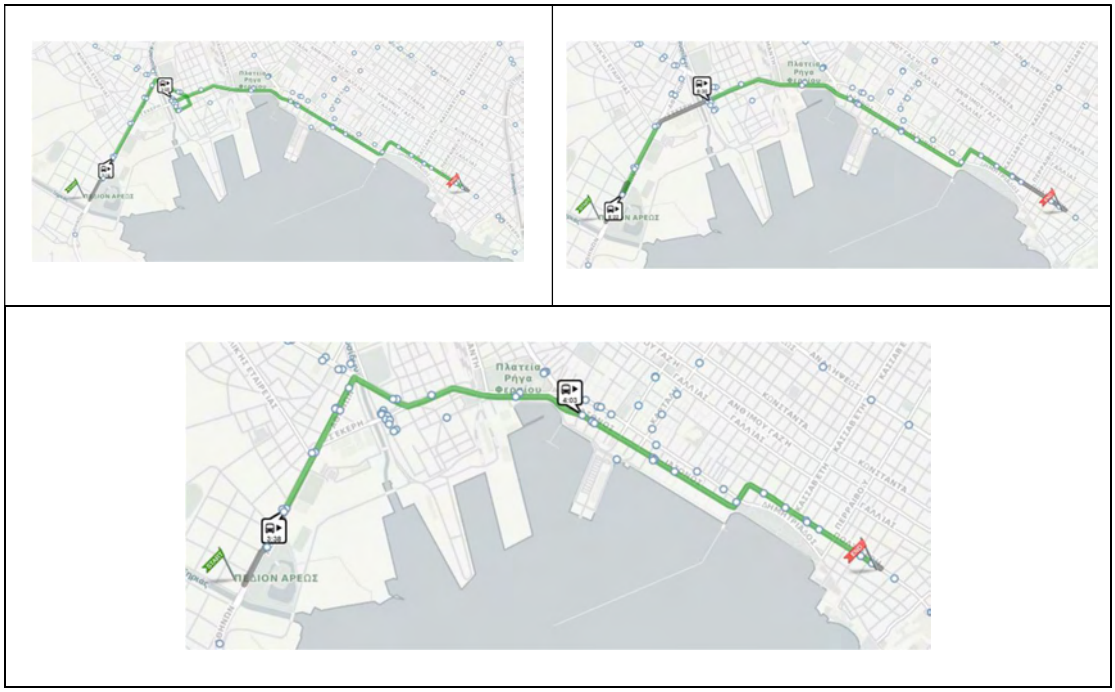

Figure 6: An example of graphical representation of a MMRP system. Specifically, those screenshots are from the OTP user interface.

Now, what is really worth mentioning is the *event* based modelling. While, not so common in other route planning topics, the timetable of a transportation service can be viewed as matrix of events occurring at certain time. Most of the graph approaches that we will present, use vertices to represent events and not geographical locations. Of course, the event describes an *arrival* or *departure* at a geographical location but not the location itself. In the form of graphs, such ideas are implemented in parallel graphs that *expand* in time or have arcs that are *time-dependent.* But, this matter will be discussed further later on in the paper.

Another turning point for how the MMRP problem has been viewed by researchers and the industry was around 2004 when Google introduced the General Transit Feed Specification (GTFS) [25], a framework on how transport operators should homogenize and share information about their network and timetables. A decade after the point of its invention, the GTFS has been the tool that most of companies use in order produce their data. The byproduct of those changes is that the availability of data has rocketed and the applications for the MMRP has gone up, in similar fashion. By 2017, much software has been developed in order to build data according to the GTFS and other programs like ArcGIS, software used for managing spatial data, have incorporated special modules to handle GTFS data. Finally, platforms have been developed where GTFS data from all over the world are available to every user. For example, researchers from Indonesia can have access to data from Greece and Europe and vice versa.

In the context of our work we developed software that can combine geo-spatial data into GTFS by combining them with timetables. That happened because we needed a way to test

our algorithms with GTFS from our region, Volos. In Section 3.1 of the thesis we present more information about the GTFS and how they work and in Section 3.1 we present the software and its functionality.

3.1 How the GTFS work

Let's proceed to give a little more insight on the GTFS. Every public transport operator has a schedule, based on which the whole organization operates. Thus, the importance of that schedule and of its proper presentation is very important. Especially, when the data are publicly available and about to be used by 3rd parties and developers all over the world, the content needs to be polished and crystal clear. For that reason, the GTFS adopts a very simple layout for the "feed", which is the main file of the GTFS. A GTFS feed is a zipped file(.zip), which contains a series of text files(.txt). The number of the files to be included by the organization is not specific, but varies, because some of the files are optional to include. For example, the *fare_attributes.txt* file is used to store extra information on the prices of the ticket to use the service, but is characterized as *optional* by the GTFS and data providers can exclude it, if it serves them better. Below, we provide a list with the files that should be included in the zip file for the feed to be considered valid. Next to the name, there is a column stating if the file is *optional* or *required* and in the third column there is a short explanation of what is included in the file:

agency.txt	Required	One or more transit agencies that provide the data in this feed.
stops.txt	Required	Individual locations where vehicles pick up or drop off passengers.
routes.txt	Required	Transit routes. A route is a group of trips that are displayed to riders as a single service.
trips.txt	Required	Trips for each route. A trip is a sequence of two or more stops that occurs at specific time.
stop_times.txt	Required	Times that a vehicle arrives at and departs from individual stops for each trip.
calendar.txt	Required	Dates for service IDs using a weekly schedule. Specify when service starts and ends, as well as days of the week where service is available.
calendar_dates.txt	Optional	Exceptions for the service IDs defined in the calendar.txt file. If calendar_dates.txt includes ALL dates of service, this file may be specified instead of calendar.txt.
fare_attributes.txt	Optional	Fare information for a transit organization's routes.
fare_rules.txt	Optional	Rules for applying fare information for a transit organization's routes.

shapes.txt	Optional	Rules for drawing lines on a map to represent a transit organization's routes.
frequencies.txt	Optional	Headway (time between trips) for routes with variable frequency of service.
transfers.txt	Optional	Rules for making connections at transfer points between routes.
feed_info.txt	Optional	Additional information about the feed itself, including publisher, version, and expiration information.

Figure 7: Files that should be contained in every feed of data that complies to the General Transit Feed Specification(GTFS)

With every file containing different information, each of the files is expected to be a Comma Separated Values file (CSV file, .csv). CSV files are used to represent matrices and are essentially text files that use commas in each row to break down each line to column fields. In other words, one can think of the files inside a GTFS feed as matrices. Their size in the computer memory vary from a few kilobytes, in files like the *agency.txt* file, up to a few gigabytes for files such as the *stop_times.txt*. Let's not forget to mention that their debugging can be very difficult because some of the files contain hundreds of thousands of lines in many cases and tools need to be used for their validation [26].

3.2 Modelling and solution approaches to the MMRP problem

For MMRP in public transit networks, we will not include separate sections for the modelling and then for the algorithms that are used to search the set of candidate solutions. Instead, we will sum up the modeling and the solution approaches for graphs in 3.2.1 Section and the MILP part of the modeling and solutions in 3.2.2. The reason for that is, most of the modeling approaches are based to what we already have referred to in Section 2, and we believe it is needless to re-state many of the ideas.

3.2.1 Graph based approaches

Graph approaches like CH, hierarchical approaches, arc flags and geometric containers have been used with slight modifications for the multi modal case. Those, as one could claim, "traditional" approaches, that served the route planning systems for many years, do not perform very well for public transit networks because the schedule based nature of the services produced big datasets, which, in turn, created big graphs. Bigger graphs take a lot more time to search.

But let's take some time to discuss two ideas that are used extensively in the modeling of the MMRP problem. The modelling of the problem is based on either of them and they can

be considered as opposites of one another. The ideas concern the way we see events and how we include them in the models. The first is the *time-expanded* approach, where we create an expanded graph that contains vertices for every event that will possibly occur in time. To be more specific, every node represents a *departure* or *arrival* at geographical location in time. In this way, we have a graph that contains multiple nodes for a single geographical location and the proper connections between them with the use of arcs. On the contrary, *time-dependent* graphs are graphs that are based around the idea that every node still represents a geographic location, but we use arcs to symbolize the event and the dependence on the passing of time. So, in a time-dependent approach graph, the number of nodes is rather small compared to a time-expanded graph, but, the number of arcs is greater. As we will see later on, in the literature there is not clear distinction between which way of approaching the problem is better, but the time-dependent graph is expected to perform better. Below, we quote an image from the journal article of Pyrga et al. [27], that efficiently depicts the differences between a time expanded and time dependent approach. In the same article by Pyrga et al., the reader can read about comparisons of implementations of the above concepts and the results when tested on networks in France and Europe.

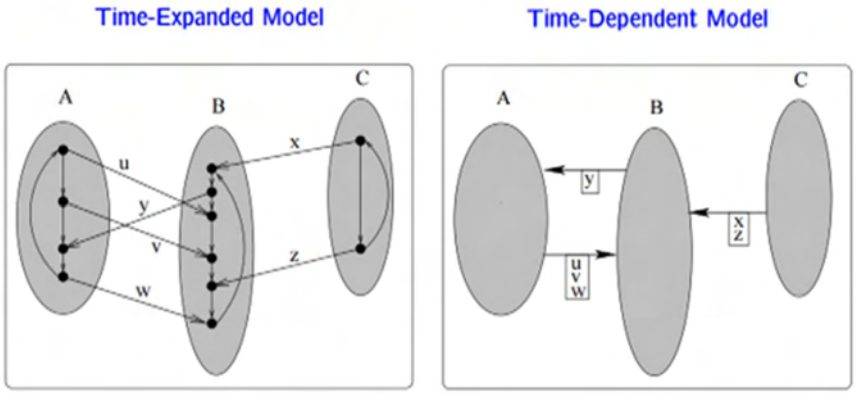

Figure 8: The difference between the time-expanded (on the left) and the time-dependent (on the right) models. If A, B and C are geographical locations on a map, we see that the time expanded creates multiple vertices along each location, whereas the time-dependent creates multiple arcs (one arc with multiple labels, each label for each event). Image taken from [27]

Subsequently, a reference will be made to the techniques that have been used and succeeded in MMRP problems. As seen in the work of Goldberg et al. [28] and Antsfeld et al. [29], researchers managed to apply *Transit Node Routing (TNR)* on the multi-modal and multi-criteria scenario route planning case. TNR, as introduced in Section 2, was one of the first approaches that were 'brought' from route planning to MMRP. But, although many advancements were introduced to TNR and other Dijkstra-based approaches, there was a need for faster computations. Graph approaches, when used for big datasets, led to complex implementations with vast amounts of precomputation. No matter the size and how advanced the precomputations are, the need to find optimal paths from one continent to another among multiple modes of transport led to the development of superior methods. Those newly introduced methods are not even based on graphs. The shiniest examples are RAPTOR algorithm [30] and Connection Scan Algorithm (CSA) [31]. The algorithms are similar in the way the operate, yet, they provide a unique perspective on how to deal with MMRP. The latter, CSA, is an approach based upon the "connection" in a timetable that describes a service

of an operator. It is focused on the earliest arrival problem, a variant of the shortest path problem. The precomputation part of the approach assembles the timetables connections into a single array and sorts them by departure time. Then in the solution space search phase, given a starting stop and the departure time, a set of labels is initiated for each stop, keeping track of the earliest arrival time at each stop. The researchers claim that a linear search of the table produces faster results than any other method even in real case scenarios, when the algorithm should consider more than one modes, meaning multiple timetables, and other criteria such as traffic.

Now moving on to RAPTOR algorithm, the work of Delling et al. was disruptive because, as far as we are concerned, before its introduction there were no mentions of non-Dijkstra approaches to tackle such problem. In its basic version, the algorithm solves the problem for two criteria, number of transfers and earliest arrival to a possible destination stop. Notice that RAPTOR is not a graph approach and stops do not correspond to vertices. RAPTOR works in three stages. At the first initiates, the labels for each node of the network that correspond to earliest arrival time at each node, and then, in the second stage it searches the timetables, based on the routes and finds which stop can be reached by which trip of those routes. Depending on the connectivity of the stops and the trips, the labels take values accordingly to the earliest arrival times at each node. In step three it incorporates the walking, as a mode of transport, that is more universal and connects the rest of the connections to one unified path. For more information on the algorithm please see [30], where the authors explain in depth the functionality of the method and introduced variants of the initial proposition (McRAPTOR and rRAPTOR) for real time traffic data with very efficient results. RAPTOR approaches have gained significant popularity because they enable developers to make use of parallelization methods that speed them up even further. OTP software is working on an implementation of RAPTOR and intents to make it its main algorithm for production systems.

4 Software developed in order to tackle the problem of creating data according to the GTFS

In our quest to come up with better solutions for the journey planning problem we encountered a more difficult problem rather than developing suitable algorithms for the different variants of the problem. Although, libraries and packages of software exist on the world wide web, they are not able to produce sufficiently homogenized data for our applications.

At first, our team didn't search for GTFS specific applications or data. We experimented with several Geographic Information Systems (GIS) and to tried to find sustainable ways to gather and organize data. Data for the city of Volos were created using only the Quantum GIS (QGIS) software [32]. Next, we experimented with other solutions that preexisted, such as the *gtfs-editor* [33]. But our team needed a way to include the shapes of the routes, a feature that didn't exist in none of the packages. Notice, that Section 3.1, that we give an explanation of the GTFS, the *shapes.txt* file that is used for describing the visual aspect of a route is an *Optional* file and as a result many organizations that produce GTFS or software to create GTFS, don't include it in their work.

We implemented a GTFS creation software called *shp2GTFS*, that did exactly what we wanted. Essentially, it combined geospatial data that represent the stops and routes with timetable information about the times of the day that the itineraries are run by the service operator. The pseudocode was written in Python programming language [34] and had as an input shapefile(.shp) layers and turned them into GTFS files, creating all the required sub-files, plus the *shapes.txt,* the file containing the information on how to display the route. Of course, as you would have guessed by now, the software didn't just "create" the data by itself but relied on other software, making it a little less autonomous. The software also makes use of a database to store all of the required information. Database used is PostgreSQL database.

For further information on how the software works we urge the reader to read the "Introduction to the methodology of GreenYourMove project for collection, creation and homogenization of public transit network data", a guide on the process we followed, with step by step explanation of the process.

5 Solution approaches developed

In the context of the current thesis, we developed 4 theoretical solution approaches and one real-world method to compute shortest paths for the case of Volos and Volos City Busses. In this paragraph, we epigraphically refer to every approach we developed and the tools used for each one. In the first and second, we simply implemented a mathematical model and a graph-based approach to describe and solve the MMRP problem. In the third and fourth approaches, we introduce on that same models, a heuristic rule based on geometric shapes that enables faster computation of paths. Finally, approach 5 employs a heuristic solution algorithm that finds the optimal route by minimizing the walking distance that the user will need to walk. In this final approach, we developed a website to visually present the solutions.

As for the tools we used, for approaches 1 to 4 we used Python programming language. For approaches 1 and 3, the library that enables to solve the mathematical models was COIN-OR and the programming interface to Python was PuLP [35]. Also for approach 2 and 4 we used NetworkX library [36] to help us build the graphs and execute queries for optimal solutions in them. At last, for approach 5, the website, most of our work was based on Javascript programming language for the heuristic and the hosting of the application, as well as HTML and CSS for the design of the website.

5.1 Approach 1: The exact Mathematical model

For our first approach, we employed techniques used in mathematical optimization. The modelling that our work was based on is the one presented in the work of Saharidis et al. [7]. The author of this thesis was a co-author in this journal article, so we had no issue replicating the formulation and improving it. The model itself is a model of Mixed Integer-Linear Formulation meaning that includes both continuous and integer variables. But just like any other optimization problem, it has three essential components the variables, the constraints and the objective function.

Nonetheless, before we get into the actual presentation of the mathematical formulation, we would to like to remark the notion behind the decisions we made on how to formulate the problem. Out of the three components of the model, the most important ones, according to our comprehension of MILP methods, are mathematical variables. Whereas many ideas come to mind, such as constructing binary variables for a whole feasible path, taking the value of 1 when the path is the solution or 0 if not, or, constructing decision variables that represent transfers from a physical stop to another, we based the decision variables on *time-expanded* graph-like approach. Each variable represents an arrival or departure event in time. But let's not elaborate more on that right now, and let's move on to 5.1 where we will present the components of the model (objective function, decision variables and constraints) and explain what they mean for the modeler, the modeler and the solver library. We want the reader to fully understand the correspondence between mathematical objects and real-world notions.

5.1.1 Nomenclature of the problems components

In the following matrix, you can see the indices and the rest of the Nomenclature that we use in order to represent the data and the decision variables in the model to follow. In comparison to [7], we introduce a new variable called *RAT* which is used to represent the arrival time on the final node. The notion behind introducing such a variable was to enable as a more robust formulation, that was less dependent on data and computed results faster. Also, we would like to make special reference to decision variable *U*, which is used to keep track of when a node is visited in a valid path. In more detail, for every feasible path the computer may come up with, U takes the positive integer value that represents the time of departure from each node contained in the feasible path. This variable is especially useful because it allows us to form constraints that eliminates unfeasible paths.

Set & Indices	
i, j, h	Indices used to represent nodes of the network;
k	Index used to represent the modes or transport;
n	Index used to represent the trip;
y	All trips;
N	Total number of nodes;
M	Total number of modes of transport;
L	Total maximum number of trips in all available modes;
Y	Total number of all trips ($Y=M*L$).
Data	
$C_{i,j,k}$	Cost moving from node i to node j with mode k;
$TT_{i,j,k}$	Travel Time moving from node i to node j with mode k;
$ToD_{i,j,y}$	Departure time of trip n with mode k from node i to j. If no such transfer exists this parameter's value is initialized to 0;
S	Departure station;
T	Arrival station;
a	Weight coefficient for environmental cost in the objective function;
b	Weight coefficient for time in the objective function;
c	Weight coefficient for arrival time in the objective function;
DT	Departure time of the user from the starting point.
Decision variables	
$X_{i,j,y}$	Binary variable used to represent whether a transfer is made from i to j with trip y ($k*n$, mode k, trip n). This decision variable is equal to 1 when the transfer is made and 0 when it is not;
U_i	Non-negative continuous variable used to represent the departure time from i. If the journey does not depart from i, U_i is equal to 0;

RAT	Real Arrival Time at target node *T*.

Figure 9: Nomenclature of the mathematical model for approach 1

5.1.2 Constraints of the mathematical model

The constraints of this model are presented and explained as follows:

$$\sum_{j=1}^{N}\sum_{y=1}^{Y} X_{S,j,y} = 1 \quad (1)$$

Constraint (1) ensures that the sum of X variables representing the departure from the starting node will be equal to 1. In that way, every feasible path includes the departure from S for sure. Later with the addition of other constraints we ensure that the departure from S is the first departure also.

$$\sum_{i=1}^{N}\sum_{y=1}^{Y} X_{i,T,y} = 1 \quad (2)$$

In the same manner, constraint (is used to make sure a connection between any point and target stop *T* is made. Or in other words, the arrival from any node to the last one T happens exactly once. Similarly, to (1), later one we introduce a constraint to the model to ensure that it is also the last connection to happen.

$$\sum_{j=1}^{N}\sum_{y=1}^{Y} X_{i,j,y} \leq 1, \forall i \neq T \quad (32)$$

Constraint (32) guarantees that the journey that is produced goes through each node at most once. Note that any other path that maybe calculated, during the enumeration of the solution space by the solver, that does not comply to the constraints, is considered infeasible and is rejected as a solution.

$$\sum_{i=1}^{N}\sum_{y=1}^{Y} X_{i,h,y} - \sum_{j=1}^{N}\sum_{y=1}^{Y} X_{h,j,y} = 0, \forall h \neq S,T \quad (43)$$

Constraint (43) ensures that whatever node of the network we visit, we have to leave from it too, except for the first and the last node. This constraint ensures a continuity in the model, and in an essence, makes sure that the algorithm will not return any feasible paths that stop at some random stop of the transit network, and for example, depart from another, without having an arrival event at latter node.

$$\sum_{i=1}^{N}\sum_{y=1}^{Y} X_{i,S,y} = 0 \tag{54}$$

Constraint (54) guarantees that our model will deliver a journey that once it has departed from *S*, it will never go through *S* again. In other words, no arrival at node S is feasible and should not be included in the calculate path.

$$\sum_{j=1}^{N}\sum_{y=1}^{Y} X_{T,j,y} = 0 \tag{6}$$

Constraint (6) makes sure that once the journey reaches the target node it never departs from it again. Similar to (5), no departure event is allowed from the last node of the network.

$$U_i - \sum_{j=1}^{N}\sum_{y=1}^{Y} \overline{ToD_{\iota,J,y}} * X_{i,j,y} = 0, \forall i \neq T \tag{75}$$

Constraint (75) is used to connect the values of the variables U_i with the values of the variables $X_{i,j,y}$. Variable U, takes the values of ToD when event X in is included in the path.

$$RAT - \sum_{i=1}^{N}\sum_{k=1}^{M}\sum_{n=1}^{L} X_{i,T,y} * (\overline{ToD_{\iota,T,y}} + \overline{TT_{\iota,T,k}}) = 0 \tag{86}$$

Constraint (86) is used to connect the value of the variable RAT with the values of the variables $X_{i,j,y}$. RAT takes the value of the arrival at node T due to (8).

$$\sum_{i=1}^{N}\sum_{y=1}^{Y}(X_{i,h,y} * \overline{ToD_{i,h,y}}) + \sum_{i=1}^{N}\sum_{k=1}^{M}\sum_{n=1}^{L}(\overline{TT_{i,h,k}} * X_{i,h,k,n}) - \sum_{j=1}^{N}\sum_{y=1}^{Y}(X_{h,j,y} * \overline{ToD_{h,j,y}}) \leq 0, \forall h \neq S,T \quad (97)$$

One could claim that (97) is the most important constraint of the problem. While constraint (4) is used to ensure continuity between X variables, expression (9) guarantees that variables *U* take the values we want them to take and, in a way, ensures the continuity in the domain of time.

$$U_S \geq \overline{DT} \quad (108)$$

Constraint (108) ensures that the departure from source *S* should be later than or equal to the desired *DT* of the user.

5.1.3 Objective function

Finally, below is the objective function of the problem. It is used to describe a multi-criteria problem, because, as you can see by reading the expression, we don't only consider time but also the cost of the transaction. In our example, the cost is the financial cost from moving from *I* to *j*, but It can be any cost, from environmental to other types of cost.

$$Min \quad \sum_{i=1}^{N}\sum_{j=1}^{N}\sum_{y=1}^{Y}\left(\bar{a} * \overline{C_{\iota,J,k}} + \bar{b} * \overline{TT_{\iota,J,k}}\right) * X_{i,j,y} + c * RAT \quad (9)$$

where *y=k*M+n*

This approach is encoded into pseudo code in Python and PuLP library and then solved by COIN-OR. For more detailed results on how the method performed you can check Section 5.5 and for the code we encourage the reader to check the Appendix and, specifically, Section 7.1.1 which refers to the first approach.

5.2 Approach 2: The heuristic approach

Now, as for the second approach that we decided to include in this thesis, it is a heuristic graph approach. We needed to implement it because of its importance in the literature and the reportedly good computational results that have been found. No matter where you look in the literature, there is a wealth of information on graphs and graph algorithms used in route

planning and MMRP. However, a graph approach that could tackle efficiently our problem could be very hard to build. Nonetheless, with a simple search on a world wide web search engine, one can come across Python libraries, or in other words bundles of software, upon which a researcher can base his work on. Tensorflow, scipy, igraph and NetworkX(which we used) are all very well structured projects with lots of documentation in order to get to know how to work with them. We chose the NetworkX library to work with, because of its simplicity and its robustness. Another important factor is that it has been used in similar implementations of algorithms. It allowed us to spent less time on 'reinventing the wheel', meaning coming up with a Dijkstra implementation, and focus our efforts on building the graph according to our GTFS data.

For this approach, we built a *time-expanded* graph based on our dataset for the Volos City Busses. Every *physical* bus stop has been depicted in the graph by several vertices, one for each time the bus went through the stops. Then, we serialize the graph to our hard drive, so every time that we needed to run queries on the graph, it wouldn't be needed to be built from the beginning. Next, we tested how much time the method needed to compute paths between random vertices of the graph. Also, we checked the results of the searches on the graph for the 'quality' of the routes returned. We did so, because for the searching for optimal solutions we made use of the Dijkstra and Bi-directional Dijkstra algorithms (both algorithms described in Section 2 & 3 of this thesis). Those algorithms are heuristic and do not extensively search the solution space which in some cases returns solutions with an *optimality gap*. By checking the supposed "quality", we searched for cases where the solution was not *high quality*, or in other words, was not the globally optimal. That was not proven to be the case and approach 2 was producing solutions of equal quality to exact methods.

As mentioned in Sections 2 & 3, where we describe the algorithms and provide a theoretical review on the subject of route planning, Dijkstra is an algorithm that when applied to a graph return the shortest path from one vertex to all. However, in most implementations of the algorithm, Dijkstra and priority queues have been intertwined. Priority queues are data structures that enable heuristics to be incorporated in the algorithm. By its use, Dijkstra can search the graph in a more direct way and check for optimal solutions in search spaces where it makes better sense. In our approach, and because the network is small in size, there was no optimality gap, because the priority queues used were describing all possible solutions and no meaningful choices were excluded from the search space. In other words, although the algorithm didn't not search the whole solution space it returned the global optimal solution to the MMRP problem. In many other cases, the priority queue is implemented using different rules such as those we mention in Section 2. It is very common for approaches like Arc Flags, TNR and others to not reach globally optimal solutions, because the rules the incorporate force them to search in solution spaces so small that don't include the global solution. However, the small search spaces enable them to solve big problems faster so, we can't say that heuristics are a drawback of those methods, but rather, an advantage. It's like a big tradeoff between speed of computation and optimality gap of solutions provided, and for each approach, and depending on the problem and its size, there are many different combinations of this tradeoff that serve the approach or make it worse.

5.3 Approaches 3 & 4: The hybrid approaches

In this chapter, we introduce two approaches that try to combine the best elements of two worlds. On the one hand, we try to find heuristics to cut down the initial solution space to smaller ones and then apply search methods to them. On the other hand, we try to reduce the optimality gap of solutions by employing exact tactics. We explore the capabilities and possibilities for each method, taking care each time of what attributes we compromise in order to gain others. In the first hybrid approach, approach 3, we apply our heuristics as a precomputation step and cluster the events geographically but chronologically also. In the geographic perspective, we cluster the events based on which stop they refer to in hexagonal shapes, making a grid, that looks like a beehive, to separate the stops into groups. In the second hybrid approach, named Approach number 4, we apply the same rule and precomputation on a graph, clustering the stops geographically and chronologically into several layers. Then we apply Dijkstra based algorithms to calculate solutions.

The heuristic rule is the same for both approaches and we will call it beehive heuristic, giving emphasis on its geometry-based perspective, but also the reader should keep in mind that the events or graph vertices are clustered in time horizon too. The clustering is applied on GTFS data and especially the *stop_times.txt*, whose lines can be considered as events. It works in 2 steps.

The *first step*, we call it the geographical hexagonal clustering step, and is the most important step. We divide the map into several hexagons, we use a “search and cluster” algorithm which inspects every available station on the map, and appends it to a list of the corresponding hexagon. In this manner, all points are distributed to a set of clusters which can be now accessed and searched when needed. We decided that we should use the hexagonal shape for the clustering of the map, because it is a very modern method that we have seen arising as a very interesting concept with very good results when it comes to clustering of elements with geographical attributes. We briefly mention that the difference of hexagons compared to other shapes, such as triangles or squares, is that they have a better *area of map* to *clusters* ratio. In this way, we will have less clusters for the same area, and thus, our search queries will be faster, when we need to search those clusters for results. Another important attribute of hexagons, which is the case with triangles and squares also, is the fact that hexagons perfectly fit the one next to the other covering the whole area/map and including every stop. This would be harder to do with other shaped such as circles. We provide the figure below with which we try to depict how the clustering occurs.

Figure 10: City of Volos and the hexagonal grid. A visual representation of the clustering method applies geometric shapes to break down the map into smaller groups

Nonetheless, we are aware that in other applications except for our example, other shapes and other kinds of clustering with another *number of clusters* to *area of the map* ratio could perform better. Note that at this stage each hexagon and the corresponding lists that contain the points include, still, a big number of points and further clustering is needed. In other words, the clusters contain stops, not events.

Moving on the *second* step of the clustering, we divide the day into 30-minute intervals and we create time hexagons. In 24 hours we have got 48 time-hexagons, as we call them. Then we search through the initial hexagons, locate which stop is in which hexagon, identify in our dataset the different events that correspond to that stop and create the time-expanded clusters. Events are then added to the clusters. After this second step of the clustering procedure, the number of clusters grows importantly. In our implementation after the first step we had 100 hexagons and after the second step our clusters were 1633, however the number of events per cluster decreased significantly, allowing the algorithm to consider less events for the phase where we search for a solution. Let's note here that clusters with no events are pruned and every cluster is assigned an ID.

Also, the clustering that we just described is required to be executed as precomputation. Then it should be serialized in the hard drive and be loaded every time we want to run the algorithm. We mention this difference between a precomputation phase and a query phase or solution space search phase, because this how real-world MMRP systems work. Now that we have discussed how the clustering works, which is common in both hybrid approaches, let's move to the specifics of each approach and see how they differ.

5.4 Approach 3: Combination of the beehive heuristic with the MILP model

Now that we have clustered the data, we want after getting an input for the starting station, ending station and departure time, to pick the clusters that would include all the desired stations. The process that is followed to pick the right clusters of events that will make up our solution space will be presented next. First comes the geographical connection of clusters. Simply, we check in which cluster the first stop belongs to and which cluster the last stop belongs to. Then we choose a series of other cluster for each every 30-minute window according to four scenarios. The cluster of the *ending node* will be either *northeast*, *northwest*, *southeast* or *southwest* to the central point of hexagon that contains the starting point. Then, when we check that condition we move on to include all the hexagons that are in the direction of the ending hexagon. Of course, in order for this selection to happen we have to analyze with the use of many geometric patterns and formulas that exploit the features of the hexagonal grid. Also, very important for this selection of the grid is the IDs that we assign to each hexagon, as well as the property of the hexagonal grid to be able to get the IDs of the hexagons that we want by adding specific numbers to the ID of the current hexagon that we work with. For more information on how this method works we prompt the readers to check the Appendix to see the specifics of the code. The result from this "direction-discovery" method is that for every 30-min set of clusters we have the ID of the cluster that will contain the stops/events we want. Then, we only need to get the corresponding set of clusters for the different layers of 30-min groups of clusters. We copy the events of those clusters to a central list, so that all events are in the list. Let's call it the *final-list.*

A question that arises is if in that list, all the required events exist so that when they are translated into a model, they will be able to be combined into a feasible path. When we tested that case, it proved that sometimes it worked and sometimes not. The method of choosing the right directions is not a method that can work by itself. So, in order for the approaches to get a list of events that would guarantee that we can calculate the optimal solution for the problem, we added a second step that would update the *final-list* after doing the following:

1) Check one by one the events in existing in the final list.

2) Acquire the routes of each event. If multiple events belong to the same route, we make sure we include every route only once.

3) Go back to the *stop_times.txt* file and get all the events that belong to trips that belong to the routes of the initial final list. Put them in a list.

4) Prune that list with all the trips according to desired Departure times and desired Arrival times.

5) Name that pruned list of events the new *final-list* and feed it to the program that searches for a solution.

After all these procedures, we can provide subset of the initial GTFS and check the solution space for optimal routes. After, turning the *final-list* into an adjacency matrix the

mathematical model calculates the solution just like in Approach 3. The model is presented in Section 5.1. Further discussion about the results of this method can be found at Section 6.

5.5 Approach 4: Combination of the beehive heuristic with the graph algorithms

As you may have noticed the solution space is the same for Approach 3 and Approach 4. Concerning Approach 4 and how the beehive heuristic has been applied to a graph algorithm, we only have a few things to note since most aspects of the approach are similar to aforementioned approaches. Firstly, all the beehive procedures, as analyzed above, are the same for the graphs. Secondly, there is an important point for approach 4, where after we have the *final-list* of events we need to build a graph that the algorithm is checked-up against. In Approach 3, we could just create an adjacency matrix and feed the data to the MILP model, but in Approach 4 we have to put to use NetworkX library and build a graph, which is significantly more expensive in terms of CPU cycles and computer resources. So, in addition to the points that we mentioned that we should follow, for the forth approach to work we need to follow a sixth step:

6) Build a graph on the fly based on the *final-list.*

Although the implementation was relatively simple to everything else that we introduced, the computational cost proved to significant.

5.6 Approach 5: Real world approach and Website

We get to the final approach that we developed. With our basis being the GTFS of Volos City Busses, we tested out some other heuristic rules and other criteria to see how they perform on a real system and compare them to the approaches.

The criterion for this approach is the distance walked by the user. Similarly to the other methods, given a starting stop and a destination stop, and independent of time, the heuristic method computes the closest stop of each bus route for the starting and the ending. Then it calculates the distance to the stops and adds the distance to the starting plus the distance to the ending summing them up to a single number. Then this number is compared for every bus route for the city of Volos, and the minimal is kept as the optimal solution. Then with the use of front-end web technologies we visualize the route of the optimal solution to the user.

For the calculations prior to the visualization part, we used Javascript language. We employed the Heuristic rule of the *Manhattan distance* to calculate distances between stops based on the coordinates of each stop. The *Manhattan distance* [37] rule is also known as *taxicab distance*. They are similar to the Euclidian distance between two points and their coordinates, but simpler, requiring fewer calculations. In more detail, the Manhattan distance between two points on the Euclidian plane is the simple sum of the horizontal and vertical

distance between them. The diagonal can be computed according to the Pythagorean Theorem.

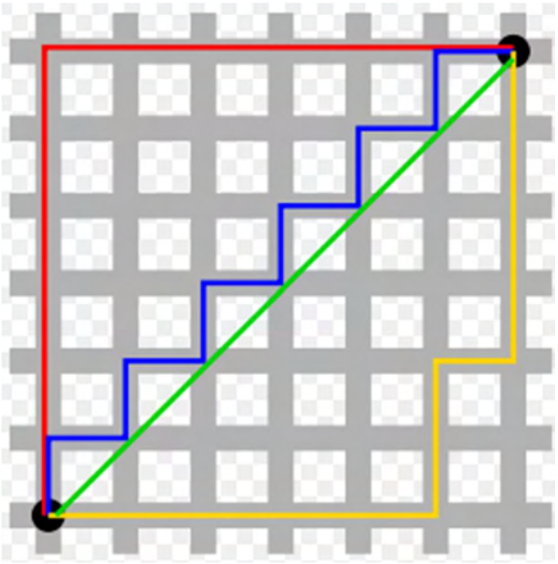

Figure 11: Grid with Pythagorean and Manhattan(taxicab) distances. Picture from [37]. The red, blue and yellow lines represent the Manhattan lines and the green represents the diagonal or Pythagorean distance.

Lastly, we developed a web application host the algorithm and a website. The web framework used is called Node.js [38] and is one of the most well-known frameworks in the industry. It rises in popularity because it enables small teams of developers to quickly deploy web applications. Speed is an important factor for start-up and spin-off companies, who played an important role in the rise of the popularity of the network. A screenshot of the website can be viewed below.

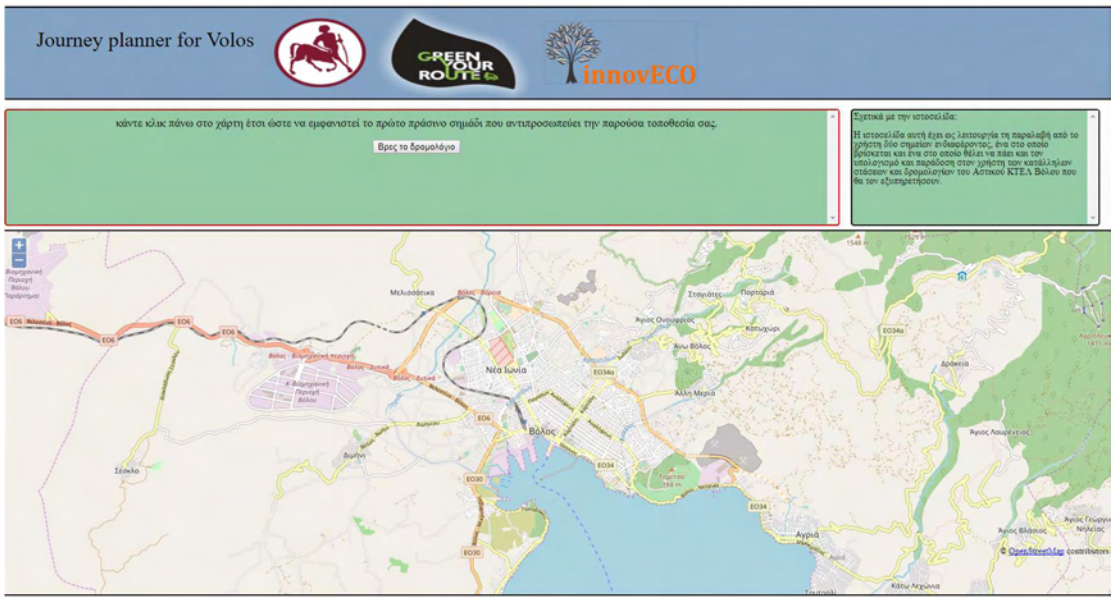

Figure 12: The website developed.

The results of this approach, as well as those of the other approaches, will be presented and discussed in the next Section.

5.7 Results

In the current Section, we will provide the results of each approach when tested against data. The data that each method was tested on were different though. Concerning the size of the problems solved, it should be mentioned that the exact approach could not solve problems with more than N=20 stops, M=11 modes of transport and L=29 trips in a reasonable time. So, the 30 random queries for these approaches were of this size. This means that these approaches do not consider data holistically. The rest of the approaches have no size limit and consider data holistically. In the results matrix, we make obvious in the second column whether a method was tested against all data of Volos or not. In the same manner, it is important to notice that not all methods require precomputations. In the second column of the matrix we let the reader know what is the case with precomputation and each approach.

Each approach was used to solve bundles of 30 queries. We did the calculations for the bundles of 30 queries 12 times for each approach, and we deleted the maximum and minimum out of those 12 executions of the bundles of 30 random queries. In this manner, we avoided the sometimes-extreme cases of executions where for random reasons (update of Windows without us knowing) may have affected the process. Other important information on the way that we executed the experiment are that we run the software that we developed on Desktop computer with the following specifications:

- CPU: AMD Phenom X2 Quad Core Processor (3.0 Ghz, 8MB)
- RAM: 12GB

The matrix below contains the results of our experiment:

Approach name	Holistically considers data/ Contains precomputation	Precomputation time (in secs)	Average CPU time for 30 queries (in secs)	Min CPU time for 30 queries (in secs)	Max CPU time for 30 queries (in secs)
Approach 1: Purely Exact	No/No	-	125.62	118.93	170.94
Approach 2: Purely heuristic	Yes/Yes	222.85	13.05	0.11	25.92
Approach 3: Bee Hive + Exact	Yes/Yes	28.89	74.32	5.86	211.49
Approach 4: Bee Hive + Graph	Yes/Yes	28.89	39.32	1.74	510.97
Approach 5: Website	No/No	-	17.20	17.01	19.05

Figure 13: Results of experiment of the thesis.

As the reader can see from the results above, it is clear that heuristic approaches perform better than exact methods in such problems. Not all of the developed methods are suited for the MMRP problem. The basic criteria to determine whether an approach is *good* or a *not so suited one* is the speed of computations and on abstract level the load on the hardware used. It clear from Figure 13 that by that criteria, heuristic approaches are way better than the rest.

An interesting result regarding the hybrid approaches is that, while the exact method, when combined with the *beehive* heuristic, was speeded up, the heuristic approach was not. We believe that is because of the on-the-fly graph construction that was needed for Approach 4. However, precomputation time for the "bee hive + graph" was less than the precomputation for the heuristic. Interestingly enough, approach 5 was proved to be suitable for the facilitation of a webservice and had especially low computation times. However, the problem is solved, as mentioned in Section 5.6 is differently.

Now concerning some thoughts and insights on improvements of the approaches, we believe that it would be interesting if we could cluster the GTFS further for the Hybrid approaches by grouping the events together by the criterion of whether they are arrival events or departure events. Also, another interesting experiment to see is how the Pythagorean rule works for computations in the online platform compared to the Manhattan distance. Alternate mathematical formulations of the problem are always interesting to see in approaches like the first and the third ones. Maybe, formulations that are based on binary variables that represent no single connections between nodes but segments of a trip (more than one connections). This kind of model would incorporate precomputations to find the paths. Finally, for the heuristic approach and the beehive heuristic, an improvement that we could see in the future is precomputing everything in the graph and storing results to the hard drive, limiting the search of the solution space to a simple table-lookup. Beehive heuristic could possibly reduce the size of saved-to-disk solutions.

6 Conclusion

On the whole, the elaboration on the topic was a very interesting and pleasant experience. The most interesting concept to grasp is the trade-offs between heuristic and exact approaches and how all the different drawbacks and advantages of the approaches are, in essence, intertwined. With our work, we aimed to provide a framework, based on which future work could be conducted.

We believe that the MMRP problem is very important to society and its research and applications have a long way to go. Although we gave emphasis on how we calculate solutions, the data will play an important role to the future of this field. Decisions of policy makers on how they should be distributed will be proven to be key. It would be extremely difficult for the field to blossom if data are kept private by organizations. The GTFS standard is a good basis for a future solution to this matter of sharing data between service operators and developers of software.

Now concerning the approaches, although the difference in the speed between the exact and graph approaches can be disheartening for researchers looking to explore exact methods, we believe that exact approaches can be used, if we apply exact precomputations to the methods. That will ensure that the big search space that is created initially from the problem formulation will be broken down in smaller sub-spaces, without excluding the different options that may be present in the network. A topic that could be researched is a mathematical model that ensures that the clustering of the events happens in a way that subspaces fed to final algorithm include all options that could be the global optima according to the criteria that the user sets.

Concluding, we would like to state that, for now, and until the subject and the ideas presented are researched further, we believe that heuristic graph approaches, because of their speed, are the *to-go* approaches for any organization that is in the process of developing web-services, or other kinds of solutions, related to the MMRP problem.

7 Bibliography

[1] Wikipedia, «Environmental impact of transport,» [Ηλεκτρονικό]. Available: https://en.wikipedia.org/wiki/Environmental_impact_of_transport.

[2] Wikipedia, «Shortest path problem,» [Ηλεκτρονικό]. Available: https://en.wikipedia.org/wiki/Shortest_path_problem.

[3] G. K. D. Saharidis, «Environmental Externalities Score: a new emission factor to model green vehicle routing problem,» *Energy Systems,* 2015.

[4] Wikipedia, «Travelling Salesman Problem,» [Ηλεκτρονικό]. Available: https://en.wikipedia.org/wiki/Travelling_salesman_problem.

[5] Hannah Bast, Daniel Delling, Andrew V. Golderg, Matthias Muller-Hannemann, Thomas Pajor, Peter Sanders, Dorothea Wagner, Renato F. Werneck, «Route Planning in Transportation Networks,» 2014.

[6] G. B. Dantzig, J. H. Ramser, «The Truck Dispatching Problem,» *Management Science,* τόμ. Vol. 6, pp. 80-91, 1959.

[7] Georgios K.D. Saharidis, Dimitrios Rizopoulos, Antonios Fragogios, Chrysostomos Chatzigeorgiou, «A hybrid approach to the problem of journey planning with the use of mathematical programming and modern techniques,» *Transportation Research Procedia,* τόμ. 24, pp. 401 - 409, 2017.

[8] Georgios K.D. Saharidis, George Kolomvos, George Liberopoulos, «Modelling and solution approach for the environmental travelling salesaman problem,» *Transport Research Arena,* 2014.

[9] C. LLC, «OpenTripPlanner,» [Ηλεκτρονικό]. Available: https://github.com/opentripplanner.

[10] «GreenYourMove project,» [Ηλεκτρονικό]. Available: https://www.greenyourmove.org/.

[11] C. LLC, «Open Trip Planner Documentation: Deployments,» [Ηλεκτρονικό]. Available: http://docs.opentripplanner.org/en/latest/Deployments/.

[12] R. Geisberger, «Contraction Hierarchies: Faster and Simpler Hierarchical Routing in Road Networks,» Institut für Theoretische Informatik Universität Karlsruhe, Karlsruhe, 2008.

[13] Ekkehard Kohler, Rolf H. Mohring and Heiko Schilling, «Fast Point-to-Point Shortest Path Computations with Arc-Flags».

[14] T. W. a. C. Z. Dorothea Wagner, «Geometric containers for efficient shortest path computation,» *ACM Journal of Experimental Algorithmics,* τόμ. 10, p. 1–30, 2005.

[15] Holger Bast, Stefan Funke, Peters Sanders, Dominik Schultes, «Fast Routing in Road Networks with Transit Nodes».

[16] E. W. Dijkstra, «A note on two problems in connexion with graphs,» *Numerische Mathematik,* pp. 269-271, 1959.

[17] Wikipedia, «Dijkstra's algorithm,» [Ηλεκτρονικό]. Available: https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm.

[18] «pgRouting,» [Ηλεκτρονικό]. Available: http://pgrouting.org/.

[19] «GraphHopper,» [Ηλεκτρονικό]. Available: https://graphhopper.com.

[20] Wikipedia, «Simplex algorithm,» [Ηλεκτρονικό]. Available: https://en.wikipedia.org/wiki/Simplex_algorithm.

[21] H. A. Taha, Operations Research: An Introduction.

[22] COIN-OR. [Ηλεκτρονικό]. Available: https://en.wikipedia.org/wiki/COIN-OR.

[23] I. Research, «CPLEX Optimizer,» [Ηλεκτρονικό]. Available: https://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/.

[24] I. Research. [Ηλεκτρονικό]. Available: https://www.research.ibm.com/.

[25] Google, «General Transit Feed Spcification (GTFS),» [Ηλεκτρονικό]. Available: https://developers.google.com/transit/gtfs/.

[26] Google, «GTFS validation tools,» [Ηλεκτρονικό]. Available: https://developers.google.com/transit/gtfs/guides/tools.

[27] Evangelia Pyrga, Dorothea Wagner, Frank Schulz, Christos Zaroliagis, «Efficient Models for Timetable Information in Public Transportation Systems,» *Journal of Experimental Algorithmics,* 2007.

[28] Andrew P Goldberg, Renato Fonseca F. Werneck, «Computing Point-to-Point Shortest Paths from External Memory,» 2005.

[29] Leonid Antsfeld, Toby Walsh, «Finding Multi-criteria Optimal Paths in Multi-modal Public Transportation Networks using the Transit Algorithm».

[30] Daniel Delling, Thomas Pajor, Renato F. Werneck, «Round-Based Public Transit Routing».

[31] Julian Dibbelt, Thomas Pajor, Ben Strasser, Dorothea Wagner, «Intriguingly Simple and Fast Transit Routing».

[32] «Quantum Geographic Information System (QGIS),» [Ηλεκτρονικό]. Available: http://www.qgis.org/en/site/.

[33] Conveyal, «GTFS Editor,» [Ηλεκτρονικό]. Available: https://github.com/conveyal/gtfs-editor.

[34] «Python for Begineers,» [Ηλεκτρονικό]. Available: https://www.python.org/about/gettingstarted/.

[35] «PuLP,» [Ηλεκτρονικό]. Available: https://www.coin-or.org/PuLP/.

[36] «Python based NetworkX library,» [Ηλεκτρονικό]. Available: https://networkx.github.io/.

[37] «Manhattan distance,» [Ηλεκτρονικό]. Available: https://en.wiktionary.org/wiki/Manhattan_distance.

[38] «Node.js Web framework based on Javascript,» [Ηλεκτρονικό]. Available: https://nodejs.org/en/.

[39] «Rome2Rio,» [Ηλεκτρονικό]. Available: https://www.rome2rio.com/.

[40] Hannah Bast, Erik Carlsson, Arno Eigenwillig, Robert Geisberger, «Fast Routing in Very Large Public Transportation Networks using Transfer Patterns».

8 Appendix

In the Section 8 of the thesis, we attach the pseudocode developed for the approaches to the MMRP and other software programs that we developed. The code can be found in the CD-ROM attached.

** End of the thesis **